Lecture 12 – Separation Logic (1)

AAA551: Programming Language Theory

Jihyeok Park

2026 Spring

Recall

PLRG

- Weakest Liberal Precondition
  - Motivation
  - Weakest Liberal Precondition
  - Example – Factorial
  - Provability and Correctness of $\mathbf{wlp}$
- Dafny
  - Example 1 – Factorial
  - Example 2 – Fibonacci
  - Example 3 – IndexOf

Contents

PLRG

1. Motivation

2. IMP with Mutation

3. Separation Logic

Contents

1. Motivation

2. IMP with Mutation

3. Separation Logic

Motivation

PLRG

Consider mutable variables x and y:

$$\{x \mapsto 1, y \mapsto -\}\ [y] := 2\ \{x \mapsto 1 \wedge y \mapsto 2\}$$

where $x \mapsto 1$ means that the variable x stores an address a that points to a heap location storing the value 1, and $[y] := 2$ is an assignment that updates the value stored at the heap location pointed to by y to 2.

Is it correct?

Motivation

PLRG

Consider mutable variables x and y:

$$\{x \mapsto 1, y \mapsto -\}\ [y] := 2\ \{x \mapsto 1 \land y \mapsto 2\}$$

where $x \mapsto 1$ means that the variable x stores an address a that points to a heap location storing the value 1, and $[y] := 2$ is an assignment that updates the value stored at the heap location pointed to by y to 2.

Is it correct? **No!** The assignment $[y] := 2$ may change the value of x.

Consider the following state $\langle h, \rho \rangle$:

Heap	$h = \{a \mapsto 1\}$
Environment	$\rho = \{x \mapsto a, y \mapsto a\}$

After the assignment $[y] := 2$, the state becomes $\langle h', \rho \rangle$:

Heap	$h' = \{a \mapsto 2\}$
Environment	$\rho\ = \{x \mapsto a, y \mapsto a\}$

Thus, x points to a heap location that stores the value 2.

Motivation

PLRG

If we can **separate** the heaps for x and y, then we can prove the correctness of the triple:

$$\{x \mapsto 1 * y \mapsto -\}\ [y] := 2\ \{x \mapsto 1 * y \mapsto 2\}$$

where $\phi * \phi'$ means that ϕ and ϕ' hold for **disjoint** parts of the heap.

Motivation

PLRG

If we can **separate** the heaps for x and y, then we can prove the correctness of the triple:

$$\{x \mapsto 1 * y \mapsto -\}\ [y] := 2\ \{x \mapsto 1 * y \mapsto 2\}$$

where $\phi * \phi'$ means that ϕ and ϕ' hold for **disjoint** parts of the heap.

Assume that the variables x and y are defined in disjoint parts of the heap, and x stores the value 1 while y stores an arbitrary value (denoted by $-$).

Motivation

PLRG

If we can **separate** the heaps for x and y, then we can prove the correctness of the triple:

$$\{x \mapsto 1 * y \mapsto -\} \; [y] := 2 \; \{x \mapsto 1 * y \mapsto 2\}$$

where $\phi * \phi'$ means that ϕ and ϕ' hold for **disjoint** parts of the heap.

Assume that the variables x and y are defined in disjoint parts of the heap, and x stores the value 1 while y stores an arbitrary value (denoted by $-$).

Then, after the assignment $[y] := 2$, the variable x still stores the value 1, and the variable y stores the value 2.

Motivation

PLRG

If we can **separate** the heaps for x and y, then we can prove the correctness of the triple:

$$\{x \mapsto 1 * y \mapsto -\}\ [y] := 2\ \{x \mapsto 1 * y \mapsto 2\}$$

where $\phi * \phi'$ means that ϕ and ϕ' hold for **disjoint** parts of the heap.

Assume that the variables x and y are defined in disjoint parts of the heap, and x stores the value 1 while y stores an arbitrary value (denoted by $-$).

Then, after the assignment $[y] := 2$, the variable x still stores the value 1, and the variable y stores the value 2.

We call the logic that allows us to reason about disjointness of heap locations **Separation Logic**.

Contents

1. Motivation

2. IMP with Mutation

3. Separation Logic

IMP with Mutation

PLRG

Let's define a simple imperative language with mutation.

$$
\begin{array}{llll}
\text{Expressions} & e & ::= & n \mid x \mid e \texttt{ + } e \mid e \texttt{ * } e \mid e \texttt{ < } e \mid \texttt{true} \mid \texttt{false} \\
\text{Statements} & s & ::= & \texttt{skip} \mid x \texttt{ := } e \mid s \texttt{;} s \\
 & & & \mid \texttt{if } e \texttt{ then } s \texttt{ else } s \\
 & & & \mid \texttt{while } e \texttt{ do } s \\
 & & & \mid x \texttt{ := } \texttt{new}(e, \ldots, e) \\
 & & & \mid x \texttt{ := } [e] \\
 & & & \mid [e] \texttt{ := } e \\
 & & & \mid \texttt{free}(e) \\
\text{Values} & v \in \mathbb{V} & ::= & n \mid \texttt{true} \mid \texttt{false} \\
\text{Addresses} & a \in \mathbb{A} & = & \mathbb{Z} \\
\text{Environments} & \rho & \in & \mathbb{X} \rightharpoonup \mathbb{V} \\
\text{Heaps} & h & \in & \mathbb{A} \rightharpoonup \mathbb{V}
\end{array}
$$

We can use an integer n to represent both values and addresses.
Note that expressions do not lookup/update the heap.

IMP with Mutation

There are two kinds of **configurations**:

- Nonterminal: $\langle s, (\rho, h) \rangle$
- Terminal: $\langle \rho, h \rangle$ or `abort`

IMP with Mutation

There are two kinds of **configurations**:

- Nonterminal: $\langle s, (\rho, h) \rangle$
- Terminal: $\langle \rho, h \rangle$ or `abort`

Allocation:

$$\frac{a, \ldots, (a+n-1) \notin \mathsf{dom}(h)}{\langle x \texttt{ := new}(e_1, \ldots, e_n), \rho, h \rangle \rightarrow \langle \rho[x \mapsto a], h[a \mapsto E[\![e_1]\!](\rho), \ldots, (a+n-1) \mapsto E[\![e_n]\!](\rho)] \rangle}$$

IMP with Mutation

There are two kinds of **configurations**:

- Nonterminal: $\langle s, (\rho, h)\rangle$
- Terminal: $\langle \rho, h\rangle$ or `abort`

Allocation:

$$\frac{a, \ldots, (a+n-1) \notin \mathsf{dom}(h)}{\langle x \texttt{ := } \texttt{new}(e_1, \ldots, e_n), \rho, h\rangle \rightarrow \langle \rho[x \mapsto a], h[a \mapsto E[\![e_1]\!](\rho), \ldots, (a+n-1) \mapsto E[\![e_n]\!](\rho)]\rangle}$$

Lookup:

$$\frac{E[\![e]\!](\rho) = a \qquad a \in \mathsf{dom}(h)}{\langle x \texttt{ := } [e], \rho, h\rangle \rightarrow \langle \rho[x \mapsto h(a)], h\rangle} \qquad \frac{E[\![e]\!](\rho) \notin \mathsf{dom}(h)}{\langle x \texttt{ := } [e], \rho, h\rangle \rightarrow \texttt{abort}}$$

IMP with Mutation

There are two kinds of **configurations**:

- Nonterminal: $\langle s, (\rho, h) \rangle$
- Terminal: $\langle \rho, h \rangle$ or `abort`

Allocation:

$$\frac{a, \ldots, (a+n-1) \notin \mathsf{dom}(h)}{\langle x \texttt{ := new}(e_1, \ldots, e_n), \rho, h \rangle \rightarrow \langle \rho[x \mapsto a], h[a \mapsto E[\![e_1]\!](\rho), \ldots, (a+n-1) \mapsto E[\![e_n]\!](\rho)] \rangle}$$

Lookup:

$$\frac{E[\![e]\!](\rho) = a \qquad a \in \mathsf{dom}(h)}{\langle x \texttt{ := } [e], \rho, h \rangle \rightarrow \langle \rho[x \mapsto h(a)], h \rangle} \qquad \frac{E[\![e]\!](\rho) \notin \mathsf{dom}(h)}{\langle x \texttt{ := } [e], \rho, h \rangle \rightarrow \texttt{abort}}$$

Mutation:

$$\frac{E[\![e_1]\!](\rho) = a \qquad a \in \mathsf{dom}(h)}{\langle [e_1] \texttt{ := } e_2, \rho, h \rangle \rightarrow \langle \rho, h[a \mapsto E[\![e_2]\!](\rho)] \rangle} \qquad \frac{E[\![e_1]\!](\rho) \notin \mathsf{dom}(h)}{\langle [e_1] \texttt{ := } e_2, \rho, h \rangle \rightarrow \texttt{abort}}$$

IMP with Mutation

PLRG

There are two kinds of **configurations**:

- Nonterminal: $\langle s, (\rho, h)\rangle$
- Terminal: $\langle \rho, h\rangle$ or `abort`

Allocation:

$$\frac{a, \ldots, (a+n-1) \notin \mathsf{dom}(h)}{\langle x \texttt{ := } \texttt{new}(e_1, \ldots, e_n), \rho, h\rangle \to \langle \rho[x \mapsto a], h[a \mapsto E[\![e_1]\!](\rho), \ldots, (a+n-1) \mapsto E[\![e_n]\!](\rho)]\rangle}$$

Lookup:

$$\frac{E[\![e]\!](\rho) = a \qquad a \in \mathsf{dom}(h)}{\langle x \texttt{ := } [e], \rho, h\rangle \to \langle \rho[x \mapsto h(a)], h\rangle} \qquad \frac{E[\![e]\!](\rho) \notin \mathsf{dom}(h)}{\langle x \texttt{ := } [e], \rho, h\rangle \to \texttt{abort}}$$

Mutation:

$$\frac{E[\![e_1]\!](\rho) = a \qquad a \in \mathsf{dom}(h)}{\langle [e_1] \texttt{ := } e_2, \rho, h\rangle \to \langle \rho, h[a \mapsto E[\![e_2]\!](\rho)]\rangle} \qquad \frac{E[\![e_1]\!](\rho) \notin \mathsf{dom}(h)}{\langle [e_1] \texttt{ := } e_2, \rho, h\rangle \to \texttt{abort}}$$

Deallocation:

$$\frac{E[\![e]\!](\rho) = a \qquad a \in \mathsf{dom}(h)}{\langle \texttt{free}(e), \rho, h\rangle \to \langle \rho, h - \{a\}\rangle} \qquad \frac{E[\![e]\!](\rho) \notin \mathsf{dom}(h)}{\langle \texttt{free}(e), \rho, h\rangle \to \texttt{abort}}$$

IMP with Mutation

We use the following notation:

- $(\sigma \to^* \sigma')$ – σ' is reachable from σ in a finite number of steps.

IMP with Mutation

PLRG

We use the following notation:

- $(\sigma \to^* \sigma')$ – σ' is reachable from σ in a finite number of steps.
- $(\sigma \uparrow)$ – there exists an infinite sequence of steps starting from σ.

IMP with Mutation

PLRG

We use the following notation:

- $(\sigma \to^* \sigma')$ – σ' is reachable from σ in a finite number of steps.
- $(\sigma \uparrow)$ – there exists an infinite sequence of steps starting from σ.
- $(h \perp h')$ – the heaps h and h' are disjoint:

$$h \perp h' \iff \mathsf{dom}(h) \cap \mathsf{dom}(h') = \emptyset$$

IMP with Mutation

PLRG

We use the following notation:

- $(\sigma \to^* \sigma')$ – σ' is reachable from σ in a finite number of steps.
- $(\sigma \uparrow)$ – there exists an infinite sequence of steps starting from σ.
- $(h \perp h')$ – the heaps h and h' are disjoint:

$$h \perp h' \iff \mathsf{dom}(h) \cap \mathsf{dom}(h') = \emptyset$$

- $(h \cdot h')$ – the union of two disjoint heaps h and h':

$$h \cdot h' = h \cup h'$$

  only defined when $h \perp h'$.

Contents

1. Motivation

2. IMP with Mutation

3. Separation Logic

Separation Logic

PLRG

Let's extend the **assertion language** of Hoare Logic to reason about heaps.

$$
\begin{array}{lcll}
i, j & \in & \textbf{LVar} & \\
e & ::= & x \mid i \mid n \mid e \texttt{ + } e \mid e \texttt{ * } e & \\
\phi & ::= & \texttt{true} \mid \texttt{false} \mid e \texttt{ = } e \mid e \texttt{ < } e & \\
 & \mid & \neg\phi \mid \phi \land \phi \mid \phi \lor \phi \mid \exists i.\phi \mid \forall i.\phi & \\
 & \mid & \texttt{emp} & \text{(empty heap)} \\
 & \mid & e \mapsto e & \text{(singleton heap)} \\
 & \mid & \phi * \phi & \text{(separating conjunction)} \\
 & \mid & \phi \mathrel{-\!\!*} \phi & \text{(separating implication)}
\end{array}
$$

Separation Logic

PLRG

$$\rho, h \models_I \texttt{emp} \qquad \iff \quad h = \varnothing$$

Separation Logic

PLRG

$$\rho, h \models_I \texttt{emp} \quad \iff \quad h = \varnothing$$

$$\rho, h \models_I e \mapsto e' \quad \iff \quad h = \{E[\![e]\!](\rho, I) \mapsto E[\![e']\!](\rho, I)\}$$

Separation Logic

PLRG

$$\rho, h \models_I \texttt{emp} \iff h = \varnothing$$

$$\rho, h \models_I e \mapsto e' \iff h = \{E[\![e]\!](\rho, I) \mapsto E[\![e']\!](\rho, I)\}$$

$$\begin{aligned}\rho, h \models_I \phi * \phi' \iff\ & \exists h_1, h_2.\ (h_1 \perp h_2) \land (h = h_1 \cdot h_2) \land \\ & (\rho, h_1 \models_I \phi) \land (\rho, h_2 \models_I \phi')\end{aligned}$$

Separation Logic

PLRG

$$
\begin{aligned}
\rho, h \models_I \texttt{emp} &\iff h = \varnothing \\
\rho, h \models_I e \mapsto e' &\iff h = \{E[\![e]\!](\rho, I) \mapsto E[\![e']\!](\rho, I)\} \\
\rho, h \models_I \phi * \phi' &\iff \exists h_1, h_2.\ (h_1 \perp h_2) \land (h = h_1 \cdot h_2) \land \\
&\qquad\qquad (\rho, h_1 \models_I \phi) \land (\rho, h_2 \models_I \phi') \\
\rho, h \models_I \phi \mathrel{-\!\!*} \phi' &\iff \forall h'.\ (h \perp h') \land \\
&\qquad (\rho, h' \models_I \phi) \implies (\rho, h \cdot h' \models_I \phi')
\end{aligned}
$$

Separation Logic – Notation

PLRG

We use the following notation:

$$e \mapsto - \quad \triangleq \quad \exists v.\ e \mapsto v$$

Separation Logic – Notation

We use the following notation:

$$e \mapsto - \quad \triangleq \quad \exists v.\, e \mapsto v$$

$$e \hookrightarrow e' \quad \triangleq \quad (e \mapsto e') * \texttt{true}$$

Separation Logic – Notation

PLRG

We use the following notation:

$$
\begin{array}{lcl}
e \mapsto - & \triangleq & \exists v.\ e \mapsto v \\
e \hookrightarrow e' & \triangleq & (e \mapsto e') * \texttt{true} \\
e \mapsto e_1, \ldots, e_n & \triangleq & (e \mapsto e_1) * \cdots * ((e + n - 1) \mapsto e_n)
\end{array}
$$

Separation Logic – Notation

PLRG

We use the following notation:

$$
\begin{aligned}
e \mapsto - &\triangleq \exists v.\ e \mapsto v \\
e \hookrightarrow e' &\triangleq (e \mapsto e') * \texttt{true} \\
e \mapsto e_1, \ldots, e_n &\triangleq (e \mapsto e_1) * \cdots * ((e + n - 1) \mapsto e_n) \\
e \hookrightarrow e_1, \ldots, e_n &\triangleq (e \mapsto e_1, \ldots, e_n) * \texttt{true}
\end{aligned}
$$

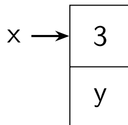
Separation Logic – Example

PLRG

$$x \mapsto 3, y$$

Separation Logic – Example

$$x \mapsto 3, y$$

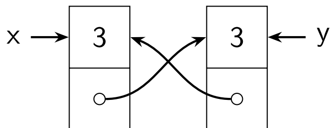


Separation Logic – Example

$$(x \mapsto 3, y) * (y \mapsto 3, x)$$

Separation Logic – Example

$$(x \mapsto 3, y) * (y \mapsto 3, x)$$



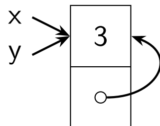
Separation Logic – Example

PLRG

$$(x \mapsto 3, y) \land (y \mapsto 3, x)$$

Separation Logic – Example

$$(x \mapsto 3, y) \land (y \mapsto 3, x)$$



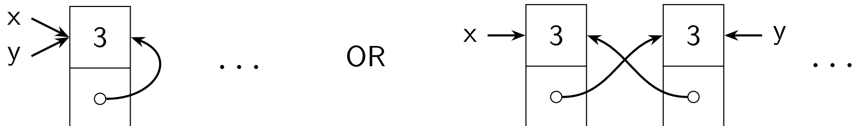
Separation Logic – Example

PLRG

$$(x \hookrightarrow 3, y) \land (y \hookrightarrow 3, x)$$

Separation Logic – Example

$$(x \hookrightarrow 3, y) \wedge (y \hookrightarrow 3, x)$$



Pure Assertions

PLRG

We call an assertion ϕ **pure** if it does not contain any heap predicate (i.e., `emp`, $e \mapsto e$, $\phi * \phi$, and $\phi \mathrel{-\!\!*} \phi$).

Pure Assertions

PLRG

We call an assertion ϕ **pure** if it does not contain any heap predicate (i.e., $\texttt{emp}$, $e \mapsto e$, $\phi * \phi$, and $\phi \mathrel{-\!\!*} \phi$).

Pure assertions only talk about the values of variables, and are independent of the heap.

For example, the following assertions are pure:

$$\begin{array}{l} \texttt{true} \\ \texttt{false} \\ x = 3 \\ x < y + 1 \\ \neg(x < y) \lor (y < z) \\ \exists i.(i > 0) \land (x = i * i) \end{array}$$

Intuitionistic Assertions

PLRG

We call an assertion ϕ **intuitionistic** it holds for a larger heap whenever it holds for a smaller heap:

$$\forall \rho, h, h'.\ (h \perp h') \land (\rho, h \models_I \phi) \implies (\rho, h \cdot h' \models_I \phi)$$

Intuitionistic Assertions

We call an assertion ϕ **intuitionistic** it holds for a larger heap whenever it holds for a smaller heap:

$$\forall \rho, h, h'. \, (h \perp h') \land (\rho, h \models_I \phi) \implies (\rho, h \cdot h' \models_I \phi)$$

Intuitionistic assertions are **preserved** when we add more heap locations.

Intuitionistic Assertions

PLRG

We call an assertion ϕ **intuitionistic** it holds for a larger heap whenever it holds for a smaller heap:

$$\forall \rho, h, h'.\ (h \perp h') \land (\rho, h \models_I \phi) \implies (\rho, h \cdot h' \models_I \phi)$$

Intuitionistic assertions are **preserved** when we add more heap locations.

For example, the following assertions are intuitionistic:

(all pure assertions)	$e \mapsto e'$
$\phi_1 \land \phi_2$	$\phi_1 \lor \phi_2$
$\forall i.\phi_1$	$\exists i.\phi_1$
$\phi * \texttt{true}$	$\texttt{true} \mathrel{-\!\!*} \phi$
$\phi_1 * \phi_2$	$\phi_1 \mathrel{-\!\!*} \phi_2$

where ϕ_1 and ϕ_2 are intuitionistic but ϕ is any assertion.

Exact Assertions

PLRG

We call an assertion ϕ **strictly exact** if it holds for exactly one heap:

$$\forall \rho, h, h'. \, (\rho, h \models_I \phi) \wedge (\rho, h' \models_I \phi) \implies (h = h')$$

All assertions built from $\mapsto$ and $*$ are strictly exact.

Exact Assertions

PLRG

We call an assertion ϕ **strictly exact** if it holds for exactly one heap:

$$\forall \rho, h, h'. \, (\rho, h \models_I \phi) \land (\rho, h' \models_I \phi) \implies (h = h')$$

All assertions built from $\mapsto$ and $*$ are strictly exact.

We call an assertion ϕ **domain-exact** if it holds for heaps with the fixed domain:

$$\forall \rho, h, h'. \, (\rho, h \models_I \phi) \land (\rho, h' \models_I \phi) \implies (\mathsf{dom}(h) = \mathsf{dom}(h'))$$

All assertions built from $\mapsto$, $*$, and quantifiers are domain-exact.

Inference Rules

PLRG

$$\frac{}{\vdash \{x = i \land \texttt{emp}\}\ x\ \texttt{:=}\ \texttt{new}(e_1, \ldots, e_n)\ \{x \mapsto e_1[x \mapsto i], \ldots, e_n[x \mapsto i]\}}$$

Inference Rules

PLRG

$$\frac{}{\vdash \{x = i \land \texttt{emp}\}\ x\ \texttt{:=}\ \texttt{new}(e_1, \ldots, e_n)\ \{x \mapsto e_1[x \mapsto i], \ldots, e_n[x \mapsto i]\}}$$

$$\frac{}{\vdash \{x = i \land e \mapsto j\}\ x\ \texttt{:=}\ [e]\ \{x = j \land e[x \mapsto i] \mapsto j\}}$$

Inference Rules

PLRG

$$\frac{}{\vdash \{x = i \wedge \texttt{emp}\}\ x\ \texttt{:=}\ \texttt{new}(e_1, \ldots, e_n)\ \{x \mapsto e_1[x \mapsto i], \ldots, e_n[x \mapsto i]\}}$$

$$\frac{}{\vdash \{x = i \wedge e \mapsto j\}\ x\ \texttt{:=}\ [e]\ \{x = j \wedge e[x \mapsto i] \mapsto j\}}$$

$$\frac{}{\vdash \{e_1 \mapsto -\}\ [e_1]\ \texttt{:=}\ e_2\ \{e_1 \mapsto e_2\}}$$

Inference Rules

PLRG

$$\frac{}{\vdash \{x = i \wedge \texttt{emp}\}\ x\ \texttt{:=}\ \texttt{new}(e_1, \ldots, e_n)\ \{x \mapsto e_1[x \mapsto i], \ldots, e_n[x \mapsto i]\}}$$

$$\frac{}{\vdash \{x = i \wedge e \mapsto j\}\ x\ \texttt{:=}\ [e]\ \{x = j \wedge e[x \mapsto i] \mapsto j\}}$$

$$\frac{}{\vdash \{e_1 \mapsto -\}\ [e_1]\ \texttt{:=}\ e_2\ \{e_1 \mapsto e_2\}}$$

$$\frac{}{\vdash \{e \mapsto -\}\ \texttt{free}(e)\ \{\texttt{emp}\}}$$

Inference Rules

PLRG

The **frame rule** allows us to reason about the part of the heap that is not modified by a statement:

$$\textsc{Frame}\ \frac{\vdash \{\phi\}\ s\ \{\phi'\} \qquad \mathsf{Modified}(s) \cap \mathsf{Free}(\phi'') = \varnothing}{\vdash \{\phi * \phi''\}\ s\ \{\phi' * \phi''\}}$$

where $\mathsf{Modified}(s)$ is the set of variables that may be modified by s and $\mathsf{Free}(A)$ is the set of free variables in A.

Inference Rules

The **frame rule** allows us to reason about the part of the heap that is not modified by a statement:

$$\textsc{Frame}\ \frac{\vdash \{\phi\}\ s\ \{\phi'\} \qquad \mathsf{Modified}(s) \cap \mathsf{Free}(\phi'') = \varnothing}{\vdash \{\phi * \phi''\}\ s\ \{\phi' * \phi''\}}$$

where $\mathsf{Modified}(s)$ is the set of variables that may be modified by s and $\mathsf{Free}(A)$ is the set of free variables in A.

Still other inference rules of Hoare Logic (e.g., consequence) are also valid in Separation Logic:

$$\textsc{Consequence}\ \frac{\vdash \phi \Rightarrow \phi_1 \qquad \vdash \{\phi_1\}\ s\ \{\phi'_1\} \qquad \vdash \phi'_1 \Rightarrow \phi'}{\vdash \{\phi\}\ s\ \{\phi'\}}$$

Example

```
{emp}
x := new(a, a);

y := new(b, b);

[x + 1] := y - x;

[y + 1] := x - y;
```

Example

$\{\texttt{emp}\}$

```
x := new(a, a);
```

$\{x \mapsto a, a\}$

```
y := new(b, b);

[x + 1] := y - x;

[y + 1] := x - y;
```

Example

$\{\texttt{emp}\}$

```
x := new(a, a);
```

$\{x \mapsto a, a\}$

```
y := new(b, b);
```

$\{(x \mapsto a, a) * (y \mapsto b, b)\}$

```
[x + 1] := y - x;


[y + 1] := x - y;
```

Example

$\{\texttt{emp}\}$

```
x := new(a, a);
```

$\{x \mapsto a, a\}$

```
y := new(b, b);
```

$\{(x \mapsto a, a) * (y \mapsto b, b)\}$

```
[x + 1] := y - x;
```

$\{(x \mapsto a, y - x) * (y \mapsto b, b)\}$

```
[y + 1] := x - y;
```

Example

$\{\texttt{emp}\}$

```
x := new(a, a);
```

$\{x \mapsto a, a\}$

```
y := new(b, b);
```

$\{(x \mapsto a, a) * (y \mapsto b, b)\}$

```
[x + 1] := y - x;
```

$\{(x \mapsto a, y - x) * (y \mapsto b, b)\}$

```
[y + 1] := x - y;
```

$\{(x \mapsto a, y - x) * (y \mapsto b, x - y)\}$

Example

$\{\texttt{emp}\}$

```
x := new(a, a);
```

$\{x \mapsto a, a\}$

```
y := new(b, b);
```

$\{(x \mapsto a, a) * (y \mapsto b, b)\}$

```
[x + 1] := y - x;
```

$\{(x \mapsto a, y - x) * (y \mapsto b, b)\}$

```
[y + 1] := x - y;
```

$\{(x \mapsto a, y - x) * (y \mapsto b, x - y)\}$

$\{\exists z.\ (x \mapsto a, z) * ((x + z) \mapsto b, -z)\}$

Next Lecture

PLRG

- Separation Logic (2)

Jihyeok Park
jihyeok_park@korea.ac.kr
https://plrg.korea.ac.kr